

1 **COMPILER DEVICE, PROGRAM, AND RECORDING MEDIUM**

2 **FIELD OF INVENTION**

3 The present invention relates to a compiler device, a
4 compiler program, and a recording medium and, more
5 particularly, to a compiler device, a compiler program, and
6 a recording medium for optimizing a program which
7 manipulates character strings.

8 **BACKGROUND OF THE INVENTION**

9 In recent years, program languages in which programmers
10 can describe intuitive operators are widely used in order to
11 improve the maintainability and robustness of programs. For
12 example, in the Java (trademark) language, a programmer can
13 easily describe a process of appending a character string to
14 a string variable, using a "+" operator.

15 Moreover, heretofore, technologies for partial
16 redundancy elimination (refer to non-patent literature 1)
17 and technologies for partial dead assignment elimination
18 (refer to non-patent literature 2) have been used in
19 compilers.

20 (Non-Patent Literature 1)

21 J. Knoop, O. Ruthing, and B. Steffen: "Lazy Code

1 Motion," In PLDI '92, 224-234, 1992.

2 (Non-Patent Literature 2)

3 J. Knoop, O. Ruthing, and B. Steffen: "Partial Dead
4 Code Elimination," In PLDI '94, 147-158, 1994.

5 In practice, a process of appending a character string
6 is compiled into a set of many processes, such as
7 reservation of a memory area and copying of data. However,
8 when a plurality of character strings are sequentially
9 appended to the same string variable, redundant reservation
10 of a memory area, redundant copying of data, and the like
11 may occur. Furthermore, these redundant instructions may
12 not be appropriately eliminated by simply adopting
13 technologies for partial redundancy elimination and partial
14 dead assignment elimination. For example, when a process of
15 appending a character string is frequently used in a program
16 for a server, the operation efficiency of the server has
17 been lowered.

18 **SUMMARY OF THE INVENTION**

19 Accordingly, an aspect of the present invention is to
20 provide a compiler device, a compiler program, and a
21 recording medium which are capable of solving the
22 above-described problems. This aspect can be achieved by

1 combinations of features described in the appended
2 independent claims. Moreover, the appended dependent claims
3 specify more advantageous concrete examples of the present
4 invention.

5 Specifically, according to a first aspect of the
6 present invention, a compiler device for optimizing a
7 program which manipulates a character string is provided.
8 The compiler device includes an append instruction detection
9 unit, a store code generation unit, and an append code
10 generation unit. The append instruction detection unit
11 detects an append instruction which appends a character
12 string to a string variable for storing a character string,
13 in the program. The store code generation unit generates,
14 as a substitute for each of a plurality of the append
15 instructions detected by the append instruction detection
16 unit, a store code for storing data of an appendant
17 character string to be appended to the string variable by
18 the append instruction into a buffer. The plurality of
19 append instructions append the character strings to the same
20 string variable. The append code generation unit generates
21 an append code for appending a plurality of the appendant
22 character strings to the string variable, at a position to
23 be executed before an instruction to refer to the string

1 variable in the program. Furthermore, a compiler program
2 and a recording medium having the compiler program recorded
3 thereon are also provided.

4 Note that the above-described summary of the invention
5 does not list all features necessary for the present
6 invention and that subcombinations of these features can be
7 also included in the present invention.

8 **BRIEF DESCRIPTION OF THE DRAWINGS**

9 For a more complete understanding of the present
10 invention and the advantages thereof, reference is now made
11 to the following description taken in conjunction with the
12 accompanying drawings, in which:

13 Fig. 1 shows a block diagram of a compiler device 10;

14 Fig. 2 shows an operation flow of the compiler device
15 10;

16 Fig. 3A shows an example of the source code of a
17 program to be optimized;

18 Fig. 3B shows the result of compiling the program;

19 Fig. 3C shows an example of a program which has
20 finished being optimized by the compiler device 10;

21 Fig. 4 shows an example where the compiler device 10
22 optimizes a program in a modified example;

1 Fig. 5A shows the result of optimizing a program in a
2 first other method;

3 Fig. 5B shows the result of optimizing a program in a
4 second other method;

5 Fig. 6 shows a block diagram of a compiler device 60;

6 Fig. 7 shows an operation flow of the compiler device
7 60;

8 Fig. 8 shows an example of a program to be optimized by
9 the compiler device 60;

10 Fig. 9 shows an example of a program obtained after a
11 semantics recovery unit 600 has detected an
12 immutable-to-mutable conversion instruction;

13 Fig. 10 shows an example of a program obtained after a
14 partial redundancy elimination unit 610 has eliminated
15 partial redundancy;

16 Fig. 11 shows an example of a program obtained after an
17 instruction elimination unit 630 has eliminated the
18 immutable-to-mutable conversion instruction;

19 Fig. 12 shows an example of a program obtained after
20 the instruction elimination unit 630 has eliminated a
21 mutable-to-immutable conversion instruction; and

22 Fig. 13 shows an example of the hardware configuration
23 of the compiler device 10 or 60.

1 **DETAILED DESCRIPTION OF THE INVENTION**

2 The present invention provide methods, apparatus and
3 systems for a compiler device, a compiler program, and a
4 recording medium which are capable of solving the
5 above-described problems. In an example embodiment, a
6 compiler device for optimizing a program which manipulates a
7 character string is provided. The compiler device includes
8 an append instruction detection unit, a store code
9 generation unit, and an append code generation unit. The
10 append instruction detection unit detects an append
11 instruction which appends a character string to a string
12 variable for storing a character string, in the program.
13 The store code generation unit generates, as a substitute
14 for each of a plurality of the append instructions detected
15 by the append instruction detection unit, a store code for
16 storing data of an appendant character string to be appended
17 to the string variable by the append instruction into a
18 buffer. The plurality of append instructions append the
19 character strings to the same string variable. The append
20 code generation unit generates an append code for appending
21 a plurality of the appendant character strings to the string
22 variable, at a position to be executed before an instruction

1 to refer to the string variable in the program.
2 Furthermore, a compiler program and a recording medium
3 having the compiler program recorded thereon are also
4 provided.

5 Hereinafter, the present invention will be described
6 through embodiments thereof. However, the embodiments below
7 are not intended to limit the invention according to the
8 appended claims. Moreover, all combinations of features
9 described in the embodiments are not always necessary for
10 solving means of the invention.

11 (First Embodiment)

12 Fig. 1 is a block diagram of a compiler device 10. The
13 compiler device 10 is, for example, a Java (trademark)
14 Just-In-Time compiler. An aspect of the compiler device 10
15 is to optimize append instructions which append character
16 strings to a string variable for storing a character string.
17 The compiler device 10 includes an append instruction
18 detection unit 100 for detecting an append instruction, a
19 store code generation unit 110 for generating a store code
20 which stores data of an appendant character string to be
21 appended to a string variable by the append instruction into
22 a buffer, a reference instruction detection unit 120 for

1 detecting a reference instruction which refers to the string
2 variable, and an append code generation unit 130 for
3 generating an append code which appends the appendant
4 character strings stored in the buffer to the string
5 variable.

6 Upon acquiring a compilation source program, the append
7 instruction detection unit 100 detects a plurality of append
8 instructions which append character strings to the same
9 string variable in the program, and then transmits a
10 detection result to the store code generation unit 110 and
11 the reference instruction detection unit 120. Based on the
12 detection result of the append instructions, the store code
13 generation unit 110 generates, as a substitute for each of
14 the plurality of append instructions in the compilation
15 source program which have been detected by the append
16 instruction detection unit 100, a store code which stores
17 data of each appendant character string to be appended to
18 the string variable by the append instruction into a buffer.
19 The store code generation unit 110 then transmits the
20 program containing the generated store codes to the append
21 code generation unit 130.

22 Based on the detection result of append instructions,
23 the reference instruction detection unit 120 detects a

1 reference instruction in the compilation source program
2 which first refers to the string variable after the
3 character strings have been appended to the string variable
4 by the plurality of append instructions, and then transmits
5 a detection result to the append code generation unit 130.
6 Based on the detection result of the reference instruction,
7 the append code generation unit 130 generates an append code
8 which appends the plurality of appendant character strings
9 to the string variable, at a position to be executed after
10 all the store codes and before the reference instruction in
11 the program received from the store code generation unit
12 110. The append code generation unit 130 then outputs, as a
13 compiled program, the program containing the generated
14 append code. Note that the compiler device 10 may execute
15 other optimization before or after the process executed by
16 each unit in Fig. 1.

17 Fig. 2 shows an operation flow of the compiler device
18 10. The compiler device 10 acquires, for example, a class
19 file containing a compilation source program in Java
20 (trademark) (S200). Next, the append instruction detection
21 unit 100 detects a plurality of append instructions which
22 append character strings to the same string variable (S210).
23 The store code generation unit 110 generates, as a

1 substitute for each of the plurality of append instructions
2 in the compilation source program which have been detected
3 by the append instruction detection unit 100, a store code
4 which stores data of each appendant character string to be
5 appended to the string variable by the append instruction
6 into a buffer (S220). The reference instruction detection
7 unit 120 detects a reference instruction which first refers
8 to the string variable after the character strings have been
9 appended to the string variable by the plurality of append
10 instructions (S230). The append code generation unit 130
11 then generates an append code which appends the plurality of
12 appendant character strings to the string variable, at a
13 position to be executed after all the store codes and before
14 the reference instruction (S240).

15 Fig. 3A shows an example of source codes of a program
16 to be optimized. The instruction on the first line assigns
17 an empty character string "" to a string variable s, which
18 is an example of string variables. Each of the instructions
19 on the second to fourth lines is an instruction which
20 concatenates an appendant character string stored in a
21 variable c with the string variable s and assigns a result
22 thereof to the string variable s, i.e., an append
23 instruction which appends the appendant character string

1 stored in the variable c to the string variable s. Note
2 that the instructions on the second to fourth lines may be
3 the same append instruction which is repeatedly executed by
4 a loop, on the source code. The instruction on the fifth
5 line is a reference instruction which first refers to the
6 string variable s after the character strings have been
7 appended to the string variable s by the append instructions
8 on the second to fourth lines.

9 Fig. 3B shows a result of compiling the program. To be
10 more specific, for example, this drawing shows an example of
11 the result of compiling the source code shown in Fig. 3A
12 into bytecode by a bytecode compiler in Java (trademark).
13 Incidentally, although the bytecode compiler actually
14 outputs bytecode in which each instruction is represented by
15 numeric data, Fig. 3B shows pseudo-code representing the
16 semantics of bytecode for convenience of explanation. Note
17 that, in Figs. 5A and 5B, and Figs. 8 to 12 to be explained
18 later, pseudo-code representing the semantics of
19 instructions will be also used for convenience of
20 explanation.

21 The instruction on the first line is an instruction
22 which converts an immutable string variable s where
23 appending a character string is not allowed, into a mutable

1 string variable sb where appending a character string is
2 allowed. For example, the instruction on the first line
3 reserves an area in memory for storing the mutable string
4 variable sb where appending a character string is allowed,
5 and copies a character string stored in the immutable string
6 variable s where appending a character string is not
7 allowed, to the mutable string variable sb. The instruction
8 on the second line appends an appendant character string
9 stored in a variable c to the mutable string variable sb.
10 Incidentally, if the variable c is not a character variable,
11 there may be cases where another immutable string variable
12 is further reserved in a further process of executing a
13 valueOf method in addition to the instruction on the second
14 line. The instruction on the third line converts the
15 mutable string variable sb into the immutable string
16 variable s. Similarly to the above, the respective
17 instructions on the fourth and seventh lines are the same as
18 that on the first line. The respective instructions on the
19 fifth and eighth lines are the same as that on the second
20 line. The respective instructions on the sixth and ninth
21 lines are the same as that on the third line. The
22 instruction on the tenth line is an instruction which refers
23 to the immutable string variable s.

1 As described above, the instruction which newly appends
2 the character string c to the immutable string variable s is
3 compiled into a plurality of instructions including area
4 reservation in memory and copying of data. According to the
5 program in Fig. 3B, the area reservation for a variable
6 which is ultimately not used is repeated by a loop. For
7 example, the area reservation for the ultimately unnecessary
8 variables sb is repeated on the first, fourth, and seventh
9 lines. Furthermore, a process of initializing the newly
10 reserved memory area is required, thus lowering efficiency.
11 Moreover, if the compiler device 10 performs dynamic memory
12 management, an unnecessary memory area is frequently
13 released by garbage collection, thus lowering the
14 efficiency.

15 Here, the mutable string variable where a process of
16 appending a character string is allowed is a variable where
17 manipulation of appending a character string is allowed by
18 the specifications of a program language. An example
19 thereof is an instance of a StringBuffer object in the Java
20 (trademark) language. Moreover, if the program language is
21 an object-oriented language, a mutable string object
22 containing the mutable string variable may have a method of
23 newly appending a character string to a character string

1 stored in the mutable string variable, é.g., an append
2 method.

3 On the other hand, the immutable string variable where
4 a process of appending a character string is not allowed is
5 a variable where manipulation of appending a character
6 string is not allowed by the specifications of a program
7 language. An example thereof is an instance of a String
8 object in the Java (trademark) language. The compiler
9 device 10 can execute a compilation process assuming that a
10 character string stored in the immutable string variable
11 does not change, thus improving the efficiency. For
12 example, the compiler device 10 does not require exclusive
13 control for accessing the immutable string variable, thus
14 enabling high-speed execution.

15 Fig. 3C shows an example of a program which has
16 finished being optimized by the compiler device 10. The
17 append instruction detection unit 100 detects, as an append
18 instruction, a combination of the instruction on the first
19 line in Fig. 3B which converts the immutable string variable
20 s into the mutable string variable sb, the instruction on
21 the second line in Fig. 3B which appends the appendant
22 character string c to the mutable string variable sb, and
23 the instruction on the third line in Fig. 3B which converts

1 the mutable string variable sb into the immutable string
2 variable s. The store code generation unit 110 then
3 generates, as a substitute for the append instruction, a
4 store code which stores data of the appendant character
5 string c into a buffer, e.g., the instruction on the second
6 line in Fig. 3C.

7 Here, the buffer is an area in memory which is
8 previously reserved for each thread executing a program.
9 Since the buffer is not used only for the execution of the
10 program shown in Fig. 3C but also for other processes on the
11 same thread, it is desired that the buffer has a size
12 sufficiently larger than the data size of a character. For
13 example, the buffer size may be 1 Mbyte or 10 Mbyte.
14 Further, the variable buf shown in Fig. 3C is a variable
15 which stores a starting address of a free space in the
16 buffer.

17 The append code generation unit 130 generates an append
18 code which appends a plurality of the appendant character
19 strings to the string variable, at a position to be executed
20 before the instruction on the seventh line which refers to
21 the immutable string variable s, e.g., on the fifth line.
22 For example, the append code generation unit 130 generates,
23 as the append code, an instruction which assigns addresses

1 of the buffer storing the appendant character strings to an
2 instance variable (s. value) of the object containing the
3 immutable string variable. As a result, the instruction on
4 the seventh line can appropriately refer to the string
5 variable obtained after the appending process has been
6 finished. It is desirable that the append code generation
7 unit 130 further generates a code which executes a process
8 of updating information indicating a free space in the
9 buffer, in case the thread executing the program shown in
10 Fig. 3C further executes another program after the program
11 shown in Fig. 3C. For example, the append code generation
12 unit 130 can appropriately notify a code to be executed
13 later of a free space in the buffer by generating an
14 instruction which updates a value of the variable buf, on
15 the sixth line.

16 As described above, the compiler device 10 can optimize
17 an append instruction which appends a character string, by
18 eliminating an instruction which reserves an area in memory
19 for a mutable string variable and an instruction which
20 copies data of the character string.

21 Incidentally, in the present embodiment, the compiler
22 device 10 is, for example, a Just-In-Time compiler which
23 compiles bytecode into executable code. Instead of this,

1 the compiler device 10 may be a bytecode compiler which
2 compiles a source program into bytecode. In this case, the
3 append instruction detection unit 100 detects, as an append
4 instruction, a "+" operator which appends character string
5 to a string variable, in a source program. Then, the store
6 code generation unit 110 generates a store code as a
7 substitute for the "+" operator. In other words, the
8 compiler device 10 may output bytecode which executes
9 substantially the same process as that of Fig. 3C by use of
10 the source program shown in Fig. 3A as input without
11 involving the program shown in Fig. 3B.

12 Fig. 4 shows an modified example where the compiler
13 device 10 optimizes a program. The compiler device 10 in
14 the present example may generate the program shown in Fig. 4
15 instead of the program shown in Fig. 3C.

16 The compiler device 10 generates the instruction on the
17 first line which initializes the buffer. The variable buf
18 has a linked list structure, and each element of the linked
19 list can store one appendant character string. The store
20 code generation unit 110 generates, as a substitute for the
21 append instruction in Fig. 3A and as a store code, the
22 instruction on the second line in Fig. 4 which stores data
23 of the appendant character string c into the buffer and the

1 instruction on the third line in Fig. 4 which acquires an
2 address indicating the next element in the linked list.
3 Similarly to the above, the store code generation unit 110
4 generates, as substitutes for the respective append
5 instructions, store codes at the fourth and sixth lines.
6 The append code generation unit 130 then generates an append
7 code which appends the plurality of appendant character
8 strings to the string variable, at a position to be executed
9 before the instruction on the ninth line which refers to the
10 immutable string variable s, e.g., on the eighth line.

11 As described above, the store code generation unit 110
12 may store, as substitutes for the plurality of append
13 instructions, combinations of the appendant character
14 strings and the addresses where the appendant character
15 strings are stored, in the form of a linked list into the
16 buffer. In this case, if the linked list is previously
17 terminated by a null pointer, the compiler device 10 can
18 detect the end of the linked list by detecting a memory
19 access violation. This makes it possible to omit an
20 instruction to check the end of the linked list in each
21 store code. Accordingly, the process can be executed at
22 high speed.

23 Fig. 5A shows a result of optimizing the program in a

1 first other method. To be more specific, in the present
2 example, an expert Java (trademark) programmer has described
3 a process of appending character strings without using "+"
4 operators in order to improve the execution efficiency of
5 character string manipulation. The instruction on the first
6 line is an instruction which reserves an area in memory for
7 storing a mutable string variable sb and copies a character
8 string stored in an immutable string variable s to the
9 mutable string variable sb. The instruction on the second
10 line appends an appendant character string stored in a
11 variable c to the mutable string variable sb. Similarly to
12 the above, the respective instructions on the third and
13 fourth lines are the same as that on the second line. The
14 instruction on the fifth line converts the mutable string
15 variable sb into the immutable string variable s. The
16 instruction on the sixth line refers to the immutable string
17 variable s.

18 As described above, according to the present example, a
19 code which appends a character string to a mutable string
20 variable is used instead of an append instruction.
21 Therefore, the program according to the present example has
22 a higher performance than the program shown in Fig. 3B.

23 Fig. 5B shows a result of optimizing the program in a

1 second other method. In the present example, a String
2 object containing an immutable string variable has a private
3 string constructor, which is not specified in the language
4 specifications of Java (trademark). With the private string
5 constructor, another String object can be newly generated,
6 and a character string obtained by concatenating a plurality
7 of String objects can be copied to the newly generated
8 String object. For example, each of the instructions on the
9 second to fourth lines is an instruction which newly
10 generates an immutable string variable and copies contents
11 of two immutable string variables to the newly generated
12 immutable string variable.

13 Thus, a conversion process into a mutable string
14 variable can be omitted. Accordingly, the program shown in
15 the present example has a higher performance than the
16 program shown in Fig. 3B. However, although an area for
17 storing an immutable string variable is reserved on each of
18 the second to fourth lines, the instruction on the fifth
19 line refers to only the immutable string variable generated
20 by the instruction on the fourth line. Therefore, the
21 program shown in the present example is still redundant.

22 On the other hand, the compiler device 10 can
23 appropriately optimize an instruction which manipulates a

1 character string, by eliminating redundant instructions to
2 reserve memory and redundant conversion processes.

3 (A Second Embodiment)

4 Fig. 6 shows a block diagram of a compiler device 60.
5 The compiler device 60 is, for example, a Java (trademark)
6 Just-In-Time compiler. An object of the compiler device 60
7 is to optimize append instructions which append character
8 strings to a string variable for storing a character string.
9 The compiler device 60 includes a semantics recovery unit
10 600, which is an example of an immutable-to-mutable
11 conversion instruction detection unit for detecting an
12 immutable-to-mutable conversion instruction which converts
13 an immutable string variable into a mutable string variable;
14 a partial redundancy elimination unit 610 for executing a
15 partial redundancy elimination process; a
16 mutable-to-immutable conversion instruction detection unit
17 620 for detecting a mutable-to-immutable conversion
18 instruction which converts the mutable string variable into
19 the immutable string variable; an instruction elimination
20 unit 630 for eliminating the immutable-to-mutable conversion
21 instruction; an inverse semantics recovery unit 640 for
22 converting the detected immutable-to-mutable conversion

1 instruction into an executable state; a measurement
2 code-containing program execution unit 642 for executing a
3 compilation source program to acquire a profile data; an
4 immutable string variable final length database 645 for
5 storing the profile data; and a mutable string variable
6 initial length specifying unit 650 for specifying the
7 initial length of the mutable string variable.

8 Upon acquiring a compilation source program, the
9 semantics recovery unit 600 detects an immutable-to-mutable
10 conversion instruction which converts an immutable string
11 variable into a mutable string variable. To be more
12 specific, for example, the semantics recovery unit 600
13 detects a combination of a plurality of instructions which
14 collectively realize a function of the immutable-to-mutable
15 conversion instruction, and converts the combination of the
16 instructions into a pseudo-instruction representing the
17 immutable-to-mutable conversion instruction. The semantics
18 recovery unit 600 then transmits the program containing the
19 pseudo-instruction obtained by the conversion to the partial
20 redundancy elimination unit 610.

21 Upon receiving the program containing the
22 pseudo-instruction from the semantics recovery unit 600, the
23 partial redundancy elimination unit 610 executes a partial

1 redundancy elimination process of moving the
2 immutable-to-mutable conversion instruction detected by the
3 semantics recovery unit 600 to the respective control flow
4 edges which merge into a single basic block before the
5 immutable-to-mutable conversion instruction. Note that
6 "moving" means changing the order of executing instructions
7 by eliminating the immutable-to-mutable conversion
8 instruction in the received program and by inserting the
9 immutable-to-mutable conversion instruction into each
10 control flow edge which merges into a single basic block.
11 The partial redundancy elimination unit 610 then transmits
12 the program obtained after the partial redundancy
13 elimination process has been executed, to the
14 mutable-to-immutable conversion instruction detection unit
15 620.

16 The mutable-to-immutable conversion instruction
17 detection unit 620 detects a mutable-to-immutable conversion
18 instruction which converts the mutable string variable into
19 the immutable string variable in the program received from
20 the partial redundancy elimination unit 610. The
21 mutable-to-immutable conversion instruction detection unit
22 620 then transmits the program received from the partial
23 redundancy elimination unit 610 with a detection result to

1 the instruction elimination unit 630.

2 The instruction elimination unit 630 has an algebraic
3 simplification unit 634 and a partial dead assignment
4 elimination unit 638. Upon receiving the program from the
5 mutable-to-immutable conversion instruction detection unit
6 620, the algebraic simplification unit 634 eliminates the
7 immutable-to-mutable conversion instruction by using an
8 algebraic simplification method. For example, the algebraic
9 simplification unit 634 eliminates the immutable-to-mutable
10 conversion instruction by using an algebraic property that
11 "if the result of converting a mutable string variable t
12 into an immutable string variable is s, then the result of
13 converting the immutable string variable s is t."
14 Specifically, the algebraic simplification unit 634
15 eliminates the immutable-to-mutable conversion instruction
16 if instructions to be executed between the
17 mutable-to-immutable conversion instruction and the
18 immutable-to-mutable conversion instruction do not modify a
19 character string stored in the mutable string variable as
20 the source variable of the mutable-to-immutable conversion
21 instruction, and if instructions to be executed between the
22 immutable-to-mutable conversion instruction and the use of
23 the mutable string variable obtained from the conversion by

1 the immutable-to-mutable conversion instruction do not
2 modify any of the mutable string variable as the source
3 variable of the mutable-to-immutable conversion instruction
4 and the mutable string variable obtained from the conversion
5 by the immutable-to-mutable conversion instruction.
6 Moreover, the algebraic simplification unit 634 causes the
7 mutable string variable as the source variable of the
8 mutable-to-immutable conversion instruction to be used as
9 the mutable string variable obtained from the conversion by
10 the immutable-to-mutable conversion instruction after the
11 immutable-to-mutable conversion instruction. Thereafter,
12 the algebraic simplification unit 634 transmits the program
13 from which the immutable-to-mutable conversion instruction
14 has been eliminated, to the partial dead assignment
15 elimination unit 638.

16 The partial dead assignment elimination unit 638
17 executes the following process as a partial dead assignment
18 elimination process. The partial dead assignment
19 elimination unit 638 moves the mutable-to-immutable
20 conversion instruction to each branch destination of a
21 branch instruction to be executed after the
22 mutable-to-immutable conversion instruction. The partial
23 dead assignment elimination unit 638 eliminates the

1 mutable-to-immutable conversion instruction if a character
2 string stored in the immutable string variable as the
3 destination variable of the mutable-to-immutable conversion
4 instruction is not referred to on each branch destination of
5 the branch instruction. Subsequently, the partial dead
6 assignment elimination unit 638 transmits the program
7 obtained after the partial dead assignment elimination
8 process has been executed, to the inverse semantics recovery
9 unit 640.

10 The inverse semantics recovery unit 640 executes a
11 conversion process of converting the mutable-to-immutable
12 conversion instruction obtained from the conversion by the
13 mutable-to-immutable conversion instruction detection unit
14 620 back into a combination of a plurality of instructions
15 which collectively realize a function of the
16 mutable-to-immutable conversion instruction. The inverse
17 semantics recovery unit 640 then transmits the program as a
18 conversion result to the mutable string variable initial
19 length specifying unit 650.

20 The measurement code-containing program execution unit
21 642 previously executes the compilation source program and
22 measures an execution status thereof in order to improve
23 efficiency of the compiled program to be generated by the

1 compiler device 60. For example, the measurement
2 code-containing program execution unit 642 measures lengths
3 of character strings actually stored in the string variables
4 to store a measurement result into the immutable string
5 variable final length database 645.

6 The mutable string variable initial length specifying
7 unit 650 receives the compilation source program from the
8 inverse semantics recovery unit 640, and receives size
9 information indicating the length of the character string
10 actually stored in the mutable string variable, from the
11 immutable string variable final length database 645. If a
12 mutable string variable is newly generated, then the mutable
13 string variable initial length specifying unit 650 sets an
14 initial length of a mutable string variable such that an
15 area equivalent to the length indicated by the size
16 information is previously reserved. This enables the
17 compiler device 60 to prevent a memory area for a mutable
18 string variable from expanding during the execution of the
19 program. Subsequently, the mutable string variable initial
20 length specifying unit 650 outputs, as a compiled program,
21 the program in which the initial length of a mutable string
22 variable has been set.

23 Incidentally, the compiler device 60 need not include

1 the measurement code-containing program execution unit 642,
2 the immutable string variable final length database 645, and
3 the mutable string variable initial length specifying unit
4 650. In this case, the compiler device 60 may use, as an
5 initial length of a mutable string variable, a value
6 previously set by the designer of the compiler device 60.

7 Fig. 7 shows an operation flow of the compiler device
8 60. The semantics recovery unit 600 detects a combination
9 of a plurality of instructions which collectively realize a
10 function of an immutable-to-mutable conversion instruction,
11 and executes semantics recovery which converts the
12 combination of the instructions into a pseudo-instruction
13 representing the immutable-to-mutable conversion
14 instruction, thereby detecting the immutable-to-mutable
15 conversion instruction which converts an immutable string
16 variable into a mutable string variable (S700). Upon
17 receiving a program containing the pseudo-instruction from
18 the semantics recovery unit 600, the partial redundancy
19 elimination unit 610 executes a partial redundancy
20 elimination process of moving the immutable-to-mutable
21 conversion instruction detected by the semantics recovery
22 unit 600 to each control flow edge which merges into a
23 single basic block before the immutable-to-mutable

1 conversion instruction (S710). The mutable-to-immutable
2 conversion instruction detection unit 620 detects a
3 mutable-to-immutable conversion instruction which converts
4 the mutable string variable into the immutable string
5 variable in the program received from the partial redundancy
6 elimination unit 610 (S720).

7 The algebraic simplification unit 634 eliminates the
8 immutable-to-mutable conversion instruction by using an
9 algebraic simplification method. For example, the algebraic
10 simplification unit 634 eliminates the immutable-to-mutable
11 conversion instruction if instructions to be executed
12 between the mutable-to-immutable conversion instruction and
13 the immutable-to-mutable conversion instruction do not
14 modify a character string stored in the mutable string
15 variable as the source variable of the mutable-to-immutable
16 conversion instruction, and if instructions to be executed
17 between the immutable-to-mutable conversion instruction and
18 the use of the mutable string variable obtained from the
19 conversion by the immutable-to-mutable conversion
20 instruction do not modify any of the mutable string variable
21 as the source variable of the mutable-to-immutable
22 conversion instruction and the mutable string variable
23 obtained from the conversion by the immutable-to-mutable

1 conversion instruction. Moreover, the algebraic
2 simplification unit 634 causes the mutable string variable
3 as the source variable of the mutable-to-immutable
4 conversion instruction to be used as the mutable string
5 variable obtained from the conversion by the
6 immutable-to-mutable conversion instruction after a position
7 where the eliminated immutable-to-mutable conversion
8 instruction existed (S730).

9 The partial dead assignment elimination unit 638
10 changes the order of executing instructions such that the
11 mutable-to-immutable conversion instruction is executed on
12 each branch destination of a branch instruction executed
13 after the mutable-to-immutable conversion instruction. The
14 partial dead assignment elimination unit 638 then executes a
15 partial dead assignment elimination process of eliminating
16 the mutable-to-immutable conversion instruction if a
17 character string stored in the immutable string variable as
18 the destination variable of the mutable-to-immutable
19 conversion instruction is not referred to on each branch
20 destination of the branch instruction (S740).

21 The inverse semantics recovery unit 640 executes a
22 conversion process of converting the immutable-to-mutable
23 conversion instruction obtained from the conversion by the

1 semantics recovery unit 600 back into a combination of a
2 plurality of instructions which collectively realize a
3 function of the immutable-to-mutable conversion instruction
4 (S750). The mutable string variable initial length
5 specifying unit 650 sets an initial length of a mutable
6 string variable based on size information indicating the
7 length of the character string actually stored in the
8 mutable string variable (S760). The length of the character
9 string actually stored in the mutable string variable is
10 previously measured by the measurement code-containing
11 program execution unit 642.

12 Fig. 8 shows an example of a program to be optimized by
13 the compiler device 60. The program shown in this drawing
14 has a basic block 800 containing the instructions on the
15 first to third lines, a basic block 810 containing the
16 instructions on the fourth to ninth lines, and a basic block
17 820 containing the instruction on the tenth line.

18 The instruction on the first line assigns an empty
19 character string "" to an immutable string variable s, which
20 is an example of immutable string variables. The
21 instruction on the second line assigns a variable min
22 indicating the initial value of an integer type variable i
23 to the integer type variable i. The instruction on the

1 third line is a branch instruction which compares the
2 integer type variable i with a variable max indicating the
3 final value of the integer type variable i, and executes the
4 basic block 820 if the integer type variable i is the
5 variable max or more, but executes the basic block 810 if
6 the integer type variable i is less than the variable max.
7 The instruction on the fourth line newly reserves an
8 area in memory for a StringBuffer object t containing a
9 mutable string variable. The instruction on the fifth line
10 appends the character string stored in the immutable string
11 variable s to the mutable string variable t. The
12 instruction on the sixth line appends the i-th character
13 string in a string array variable a to the mutable string
14 variable t. The instruction on the seventh line is a
15 mutable-to-immutable conversion instruction which converts
16 the mutable string variable t into the immutable string
17 variable s. The instruction on the eighth line increments
18 the integer type variable i. The instruction on the ninth
19 line is a branch instruction which compares the integer type
20 variable i with the variable max indicating the final value
21 of the integer type variable i, and executes the basic block
22 820 if the integer type variable i is the variable max or
23 more, but executes the basic block 810 again if the integer

1 type variable i is less than the variable max.

2 The instruction on the tenth line outputs the character
3 string stored in the immutable string variable s to standard
4 output, e.g., to a console window on a display screen.

5 Fig. 9 shows an example of a program obtained after the
6 semantics recovery unit 600 has detected an
7 immutable-to-mutable conversion instruction. The semantics
8 recovery unit 600 detects, as the immutable-to-mutable
9 conversion instruction, a combination of the instructions on
10 the fourth and fifth lines in Fig. 8, i.e., a combination of
11 the instruction which reserves a memory area used for a
12 mutable string variable and the instruction which copies an
13 immutable string variable to the mutable string variable.
14 Next, as shown on the fourth line in Fig. 9, the semantics
15 recovery unit 600 replaces the detected immutable-to-mutable
16 conversion instruction with a pseudo-instruction
17 representing an immutable-to-mutable conversion instruction
18 which converts the immutable string variable s into the
19 mutable string variable t. In other words, the semantics
20 recovery unit 600 detects a combination of a plurality of
21 instructions constituting a process of an
22 immutable-to-mutable conversion instruction, and recovers
23 the semantics of the process represented by the plurality of

1 instructions.

2 Incidentally, the semantics recovery unit 600 detects
3 the immutable-to-mutable conversion instruction by detecting
4 the instructions on the fourth and fifth lines which are
5 sequentially located, in the program. Instead of this, the
6 semantics recovery unit 600 may detect, as a
7 mutable-to-immutable conversion instruction, the
8 instructions on the fourth and fifth lines even if other
9 instructions are located between the instructions on the
10 fourth and fifth lines but if the first manipulation to the
11 newly reserved mutable string variable is the appendant of
12 an immutable character string.

13 Fig. 10 shows an example of a program obtained after
14 the partial redundancy elimination unit 610 has eliminated
15 partial redundancy. The partial redundancy elimination unit
16 610 eliminates the immutable-to-mutable conversion
17 instruction on the fourth line from the basic block 810 and
18 inserts the immutable-to-mutable conversion instructions
19 into basic blocks 830 and 840 in order to move the
20 immutable-to-mutable conversion instruction detected by the
21 semantics recovery unit 600 to each control flow edge which
22 merges into a single basic block before the
23 immutable-to-mutable conversion instruction. Incidentally,

1 to be more detailed, the partial redundancy elimination unit
2 610 may execute, as a partial redundancy elimination
3 process, substantially the same process as that of the
4 technology or the like described in non-patent literature 2.
5 For example, according to an existing technology, the
6 partial redundancy elimination unit 610 moves the
7 immutable-to-mutable conversion instruction further backward
8 under the condition that the immutable-to-mutable conversion
9 instruction is not executed before an instruction which
10 assigns a value to the variable s, which is the source
11 variable of the immutable-to-mutable conversion instruction.

12 Fig. 11 shows an example of a program obtained after
13 the instruction elimination unit 630 has eliminated the
14 immutable-to-mutable conversion instruction. The algebraic
15 simplification unit 634 eliminates the immutable-to-mutable
16 conversion instruction moved to the basic block 840, by
17 using an algebraic simplification method. Specifically, the
18 algebraic simplification unit 634 eliminates the
19 immutable-to-mutable conversion instruction in the basic
20 block 840, because, in the control flow from the basic block
21 810 to the basic block 840, instructions to be executed
22 between the mutable-to-immutable conversion instruction on
23 the sixth line and the immutable-to-mutable conversion

1 instruction on the tenth line in Fig. 10 do not modify the
2 character string stored in the mutable string variable t,
3 and because instructions to be executed between the
4 immutable-to-mutable conversion instruction and the use of
5 the mutable string variable obtained from the conversion by
6 the immutable-to-mutable conversion instruction do not
7 modify any of the mutable string variable as the source
8 variable of the mutable-to-immutable conversion instruction
9 and the mutable string variable obtained from the conversion
10 by the immutable-to-mutable conversion instruction. The
11 algebraic simplification unit 634 then causes the mutable
12 string variable as the source variable of the
13 mutable-to-immutable conversion instruction to be used as
14 the mutable string variable obtained from the conversion by
15 the immutable-to-mutable conversion instruction, after the
16 immutable-to-mutable conversion instruction (instruction on
17 the tenth line).

18 To be more detailed, the algebraic simplification unit
19 634 decides whether the character string stored in the
20 mutable string variable t is modified, using an analysis of
21 the relationship between a value definition of a variable
22 and the use of a value stored in the variable, e.g., using a
23 UD/DU chain. Further, under the condition that the mutable

1 string variable t is not accessed by the other threads, the
2 algebraic simplification unit 634 decides that the character
3 string stored in the mutable string variable t is not
4 modified. For example, the algebraic simplification unit
5 634 may use, as the condition, the fact that the program is
6 executed on a single thread. Alternatively, the algebraic
7 simplification unit 634 may confirm that pointers to the
8 mutable string variable t are not notified to the other
9 threads, by escape analysis.

10 Fig. 12 shows an example of a program obtained after
11 the instruction elimination unit 630 has eliminated the
12 mutable-to-immutable conversion instruction. The partial
13 dead assignment elimination unit 638 executes the following
14 process as a partial dead assignment elimination process.
15 The partial dead assignment elimination process 638 causes
16 the mutable-to-immutable conversion instruction on the sixth
17 line in Fig. 11 to be executed on each branch destination of
18 a branch instruction which is executed after the
19 mutable-to-immutable conversion instruction, e.g., the
20 instruction on the seventh line. In other words, the
21 partial dead assignment elimination unit 638 eliminates the
22 mutable-to-immutable conversion instruction on the sixth
23 line in Fig. 11 from the basic block 810 and moves the

1 mutable-to-immutable conversion instructions to the basic
2 blocks 840 and 850. Since a character string stored in the
3 immutable string variable s as the destination variable of
4 the mutable-to-immutable conversion instruction moved to the
5 basic block 840 is not referred to in the flow executed
6 after the basic block 840, the partial dead assignment
7 elimination unit 638 eliminates the mutable-to-immutable
8 conversion instruction moved to the basic block 840.

9 Incidentally, to be more detailed, the partial dead
10 assignment elimination unit 638 may execute, as a partial
11 dead assignment elimination process, substantially the same
12 process as that of the technology or the like described in
13 non-patent literature 1. For example, the partial dead
14 assignment elimination unit 638 moves the
15 mutable-to-immutable conversion instruction further forward
16 under the condition that the mutable-to-immutable conversion
17 instruction is not executed after an instruction which
18 refers to the variable s as the destination variable of the
19 mutable-to-immutable conversion instruction.

20 Fig. 13 shows an example of the hardware configuration
21 of the compiler device 10 or 60. The compiler device 10
22 according to the first embodiment or the modified example
23 thereof, or the compiler device 60 according to the second

1 embodiment includes a CPU peripheral unit, an input/output
2 unit, and a legacy input/output unit. The CPU peripheral
3 unit includes a CPU 1000, a RAM 1020, a graphic controller
4 1075 which are mutually connected by a host controller 1082,
5 and a display device 1080. The input/output unit includes a
6 communication interface 1030, a hard disk drive 1040, and a
7 CD-ROM drive 1060 which are connected to the host controller
8 1082 by an input/output controller 1084. The legacy
9 input/output unit includes a ROM 1010 connected to the
10 input/output controller 1084, a flexible disk drive 1050,
11 and an input/output chip 1070.

12 The CPU 1000, the graphic controller 1075 which access
13 the RAM 1020 at high transfer rates, and the RAM 1020 are
14 mutually connected by the host controller 1082. The CPU
15 1000 operates based on a compiler program stored in the ROM
16 1010 and the RAM 1020, and controls each unit. The graphic
17 controller 1075 acquires image data which the CPU 1000 or
18 the like generates on a frame buffer provided on the RAM
19 1020, and displays the image data on the display device
20 1080. Instead of this, the graphic controller 1075 may
21 include a frame buffer for storing image data generated by
22 the CPU 1000 or the like, inside the graphic controller
23 1075.

1 The input/output controller 1084 connects the
2 communication interface 1030 which is a relatively
3 high-speed input/output device, the hard disk drive 1040,
4 and the CD-ROM drive 1060 to the host controller 1082. The
5 communication interface 1030 communicates with other device
6 through a network. The hard disk drive 1040 stores a
7 compiler program and data used by the compiler device 10 or
8 60. The CD-ROM drive 1060 reads a compiler program or data
9 from a CD-ROM 1095 to provide the compiler program or the
10 data to the input/output chip 1070 through the RAM 1020.

11 Moreover, the ROM 1010, and relatively low-speed
12 input/output devices, such as the flexible disk drive 1050
13 and the input/output chip 1070, are connected to the
14 input/output controller 1084. The ROM 1010 stores a boot
15 program executed by the CPU 1000 when the compiler device 10
16 or 60 is started up, programs depending on the hardware of
17 the compiler device 10 or 60, and the like. The flexible
18 disk drive 1050 reads a compiler program or data from a
19 flexible disk 1090 to provide the compiler program or the
20 data to the input/output chip 1070 through the RAM 1020.
21 The input/output chip 1070 is connected to the flexible disk
22 1090 and various input/output devices through, for example,
23 a parallel port, a serial port, a keyboard port, a mouse

1 port, and the like.

2 A compiler program provided to the compiler device 10
3 or 60 is provided by a user in the state that the compiler
4 program is stored on a recording medium, such as the
5 flexible disk 1090, the CD-ROM 1095, or an IC card. The
6 compiler program is read from the recording medium,
7 installed on the compiler device 10 or 60 through the
8 input/output chip 1070, and executed on the compiler device
9 10 or 60.

10 The compiler program installed and executed on the
11 compiler device 10 or 60 includes an append instruction
12 detection module, a store code generation module, a
13 reference instruction detection module, an append code
14 generation module, a mutable-to-immutable conversion
15 instruction detection module, a partial redundancy
16 elimination module, an immutable-to-mutable conversion
17 instruction detection module, an instruction elimination
18 module, an algebraic simplification module, a partial dead
19 assignment elimination module, an inverse semantics recovery
20 module, a measurement code-containing program execution
21 module, and a mutable string variable initial length
22 specifying module. Operations which the compiler device 10
23 or 60 is actuated to execute by each module are the same as

1 those of the corresponding member in the compiler device 10
2 or 60 described in Figs. 1 to 12. Therefore, a description
3 thereof will be omitted.

4 The above-described compiler program and modules may be
5 stored on an external recording medium. In addition to the
6 flexible disk 1090 and the CD-ROM 1095, optical recording
7 media including DVDs and PDs, magneto-optical recording
8 media including MDs, tape media, semiconductor memories
9 including IC cards, and the like can be used as the
10 recording medium. Moreover, a storage device, such as a
11 hard disk drive or a RAM, which is provided in a server
12 system connected to a dedicated communication network or the
13 Internet, may be used as the recording medium to provide the
14 compiler program to the compiler device 10 or 60 through the
15 network.

16 As apparent from the above description, the compiler
17 device can optimize instructions which append character
18 strings.

19 Although the present invention has been described by
20 using the embodiments, the technical scope of the present
21 invention is not limited to the scope described in the
22 aforementioned embodiments. Various modifications and
23 improvements can be made to the aforementioned embodiments.

1 From the appended claims, it is apparent that aspects in
2 which such modifications and improvements are made to the
3 embodiments can be also included in the technical scope of
4 the present invention. For example, the compiler device 10
5 may further include the members of the compiler device 60.
6 In this case, the compiler device 10 may select any of the
7 functions described in the first and second embodiments,
8 based on a predetermined condition to execute a process.
9 For example, the compiler device 10 may execute a process as
10 follows: if the length of a character string can be measured
11 by the measurement code-containing program execution unit
12 642, the compiler device 10 executes the processes described
13 in the second embodiment; if the length of a character
14 string cannot be measured, the compiler device 10 executes
15 the processes described in the first embodiment. Moreover,
16 if the variance of the length of a plurality of character
17 strings measured by the measurement code-containing program
18 execution unit 642 is more than a predetermined value, the
19 compiler device 10 may execute the processes described in
20 the first embodiment.

21 According to all of the above-described embodiments and
22 modified examples, compiler devices, compiler programs, and
23 a recording medium which are shown in the respective items

1 below can be realized.

2 This includes a compiler device for optimizing a
3 program which manipulates a character string includes
4 an append instruction detection unit for detecting an
5 append instruction to append a character string to a
6 string variable for storing a character string, in the
7 program; a store code generation unit for generating,
8 as a substitute for each of a plurality of the append
9 instructions detected by the append instruction
10 detection unit, a store code for storing data of an
11 appendant character string to be appended to the string
12 variable by the append instruction into a buffer, the
13 plurality of append instructions appending the
14 character strings to the same string variable; and an
15 append code generation unit for generating an append
16 code for appending a plurality of the appendant
17 character strings to the string variable, at a position
18 to be executed before an instruction to refer to the
19 string variable in the program.

20 In some embodiments, the compiler device further
21 includes a reference instruction detection unit for
22 detecting a reference instruction which first refers to

1 the string variable after the character strings have
2 been appended to the string variable by the plurality
3 of append instructions, wherein the append code
4 generation unit generates the append code at a position
5 to be executed after all the store codes and before the
6 reference instruction.

7 In some embodiments of the compiler device, the append
8 instruction detection unit detects, as the append
9 instruction, a combination of: an instruction to
10 convert an immutable string variable in which a process
11 of appending a character string is not allowed, into a
12 mutable string variable in which a process of appending
13 a character string is allowed; an instruction to append
14 the appendant character string to the mutable string
15 variable; and an instruction to convert the mutable
16 string variable into the immutable string variable.

17 Also included is a compiler device for optimizing a
18 program which manipulates a character string includes
19 an append instruction detection unit for detecting an
20 append instruction to append a character string to a
21 string variable for storing a character string, in the

1 program; a store code generation unit for generating,
2 as a substitute for each of a plurality of the append
3 instructions detected by the append instruction
4 detection unit, a store code for storing an address in
5 memory where an appendant character string to be
6 appended to the string variable by the append
7 instruction is stored, into a buffer, the plurality of
8 append instructions appending character strings to the
9 same string variable; and an append code generation
10 unit for generating an append code for appending a
11 plurality of the appendant character strings stored in
12 a plurality of the addresses, to the string variable,
13 at a position to be executed before an instruction to
14 refer to the string variable in the program.

15 Also included is a compiler device for optimizing a
16 program which manipulates a character string includes a
17 mutable-to-immutable conversion instruction detection
18 unit for detecting a mutable-to-immutable conversion
19 instruction to convert a mutable string variable in
20 which a process of appending a character string is
21 allowed, into an immutable string variable in which a
22 process of appending a character string is not allowed;

1 an immutable-to-mutable conversion instruction
2 detection unit for detecting an immutable-to-mutable
3 conversion instruction to convert the immutable string
4 variable into the mutable string variable; and an
5 instruction elimination unit for eliminating the
6 immutable-to-mutable conversion instruction and for
7 causing the mutable string variable as a source
8 variable of the mutable-to-immutable conversion
9 instruction, to be used as the mutable string variable
10 obtained from conversion by the immutable-to-mutable
11 conversion instruction after the immutable-to-mutable
12 conversion instruction, if an instruction to be
13 executed between the mutable-to-immutable conversion
14 instruction and the immutable-to-mutable conversion
15 instruction does not modify a character string stored
16 in the mutable string variable as the source variable
17 of the mutable-to-immutable conversion instruction and
18 if an instruction to be executed between the
19 immutable-to-mutable conversion instruction and use of
20 the mutable string variable obtained from the
21 conversion by the immutable-to-mutable conversion
22 instruction does not modify any of the mutable string
23 variable as the source variable of the

1 mutable-to-immutable conversion instruction and the
2 mutable string variable obtained from the conversion by
3 the immutable-to-mutable conversion instruction.

4 In some embodiments of the compiler device, the
5 instruction elimination unit further eliminates the
6 mutable-to-immutable conversion instruction if a
7 character string stored in the immutable string
8 variable is not referred to.

9 In some embodiments of the compiler device, the
10 instruction elimination unit moves the
11 mutable-to-immutable conversion instruction to each
12 branch destination of a branch instruction to be
13 executed after the mutable-to-immutable conversion
14 instruction, and executes partial dead assignment
15 elimination for eliminating the mutable-to-immutable
16 conversion instruction if a character string stored in
17 the immutable string variable as a destination variable
18 of the mutable-to-immutable conversion instruction is
19 not referred to on each branch destination of the
20 branch instruction.

1 In some embodiments of the compiler device, the
2 immutable-to-mutable conversion instruction detection
3 unit detects, as the immutable-to-mutable conversion
4 instruction, a combination of: an instruction to
5 reserve a memory area to be used as a mutable string
6 variable; and an instruction to append a character
7 string stored in the immutable string variable to the
8 mutable string variable.

9 In some embodiments of the compiler device, the
10 compiler device further includes a partial redundancy
11 elimination unit for executing a partial redundancy
12 elimination process of moving the immutable-to-mutable
13 conversion instruction detected by the
14 immutable-to-mutable conversion instruction detection
15 unit to each control flow edge which merges into a
16 single control flow before the immutable-to-mutable
17 conversion instruction, wherein the instruction
18 elimination unit eliminates the immutable-to-mutable
19 conversion instruction, if an instruction to be
20 executed between the mutable-to-immutable conversion
21 instruction and the immutable-to-mutable conversion
22 instruction does not modify a character string stored

1 in the mutable string variable as the source variable
2 of the mutable-to-immutable conversion instruction and
3 if an instruction to be executed between the
4 immutable-to-mutable conversion instruction and the use
5 of the mutable string variable obtained from the
6 conversion by the immutable-to-mutable conversion
7 instruction does not modify any of the mutable string
8 variable as the source variable of the
9 mutable-to-immutable conversion instruction and the
10 mutable string variable obtained from the conversion by
11 the immutable-to-mutable conversion instruction, in a
12 program obtained after the partial redundancy
13 elimination process has been executed.

14 In some embodiments of the compiler device, the
15 instruction elimination unit moves the
16 mutable-to-immutable conversion instruction to each
17 branch destination of a branch instruction to be
18 executed after the mutable-to-immutable conversion
19 instruction, and executes partial dead assignment
20 elimination for eliminating the mutable-to-immutable
21 conversion instruction if a character string stored in
22 the immutable string variable as a destination variable

1 of the mutable-to-immutable conversion instruction is
2 not referred to on each branch destination of the
3 branch instruction.

4 Also included is a compiler program for optimizing a
5 program which manipulates a character string, by using
6 a computer causes the computer to function as: an
7 append instruction detection unit for detecting an
8 append instruction to append a character string to a
9 string variable for storing a character string, in the
10 program; a store code generation unit for generating,
11 as a substitute for each of a plurality of the append
12 instructions detected by the append instruction
13 detection unit, a store code for storing data of an
14 appendant character string to be appended to the string
15 variable by the append instruction into a buffer, the
16 plurality of append instructions appending the
17 character strings to the same string variable; and an
18 append code generation unit for generating an append
19 code for appending a plurality of the appendant
20 character strings to the string variable, at a position
21 to be executed before an instruction to refer to the
22 string variable in the program.

1 Also included is a compiler program for optimizing a
2 program which manipulates a character string, by using
3 a computer causes the computer to function as: an
4 append instruction detection unit for detecting an
5 append instruction to append a character string to a
6 string variable for storing a character string, in the
7 program; a store code generation unit for generating,
8 as a substitute for each of a plurality of the append
9 instructions detected by the append instruction
10 detection unit, a store code for storing an address in
11 memory where an appendant character string to be
12 appended to the string variable by the append
13 instruction is stored, into a buffer, the plurality of
14 append instructions appending the character strings to
15 the same string variable; and an append code generation
16 unit for generating an append code for appending a
17 plurality of the appendant character strings stored in
18 a plurality of the addresses, to the string variable,
19 at a position to be executed before an instruction to
20 refer to the string variable in the program.

21 Also included is a compiler program for optimizing a
22 program which manipulates a character string, by using

1 a computer causes the computer to function as: a
2 mutable-to-immutable conversion instruction detection
3 unit for detecting a mutable-to-immutable conversion
4 instruction to convert a mutable string variable in
5 which a process of appending a character string is
6 allowed, into an immutable string variable in which a
7 process of appending a character string is not allowed;
8 an immutable-to-mutable conversion instruction
9 detection unit for detecting an immutable-to-mutable
10 conversion instruction to convert the immutable string
11 variable into the mutable string variable; and an
12 instruction elimination unit for eliminating the
13 immutable-to-mutable conversion instruction and for
14 causing the mutable string variable as a source
15 variable of the mutable-to-immutable conversion
16 instruction, to be used as the mutable string variable
17 obtained from conversion by the immutable-to-mutable
18 conversion instruction after the immutable-to-mutable
19 conversion instruction, if an instruction to be
20 executed between the mutable-to-immutable conversion
21 instruction and the immutable-to-mutable conversion
22 instruction does not modify a character string stored
23 in the mutable string variable as the source variable

1 of the mutable-to-immutable conversion instruction and
2 if an instruction to be executed between the
3 immutable-to-mutable conversion instruction and use of
4 the mutable string variable obtained from the
5 conversion by the immutable-to-mutable conversion
6 instruction does not modify any of the mutable string
7 variable as the source variable of the
8 mutable-to-immutable conversion instruction and the
9 mutable string variable obtained from the conversion by
10 the immutable-to-mutable conversion instruction.

11 Also included is a recording medium having any of the
12 compiler programs recorded thereon.

13 As apparent from the above description, according to
14 the present invention, a program which manipulates character
15 strings can be optimized.

16 Although example embodiments of the present invention
17 have been described in detail, it should be understood that
18 various changes, substitutions and alternations can be made
19 therein without departing from spirit and scope of the
20 inventions as defined by the appended claims.

21 Variations described for the present invention can be

1 realized in any combination desirable for each particular
2 application. Thus particular limitations, and/or embodiment
3 enhancements described herein, which may have particular
4 advantages to the particular application need not be used
5 for all applications. Also, not all limitations need be
6 implemented in methods, systems and/or apparatus including
7 one or more concepts of the present invention. The present
8 invention can be realized in hardware, software, or a
9 combination of hardware and software. A visualization tool
10 according to the present invention can be realized in a
11 centralized fashion in one computer system, or in a
12 distributed fashion where different elements are spread
13 across several interconnected computer systems. Any kind of
14 computer system - or other apparatus adapted for carrying
15 out the methods and/or functions described herein - is
16 suitable. A typical combination of hardware and software
17 could be a general purpose computer system with a computer
18 program that, when being loaded and executed, controls the
19 computer system such that it carries out the methods
20 described herein. The present invention can also be
21 embedded in a computer program product, which comprises all
22 the features enabling the implementation of the methods
23 described herein, and which - when loaded in a computer

1 system - is able to carry out these methods.

2 Computer program means or computer program in the
3 present context include any expression, in any language,
4 code or notation, of a set of instructions intended to cause
5 a system having an information processing capability to
6 perform a particular function either directly or after
7 conversion to another language, code or notation, and/or
8 reproduction in a different material form.

9 Thus the invention includes an article of manufacture
10 which comprises a computer usable medium having computer
11 readable program code means embodied therein for causing a
12 function described above. The computer readable program
13 code means in the article of manufacture comprises computer
14 readable program code means for causing a computer to effect
15 the steps of a method of this invention. Similarly, the
16 present invention may be implemented as a computer program
17 product comprising a computer usable medium having computer
18 readable program code means embodied therein for causing a a
19 function described above. The computer readable program
20 code means in the computer program product comprising
21 computer readable program code means for causing a computer
22 to effect one or more functions of this invention.
23 Furthermore, the present invention may be implemented as a

1 program storage device readable by machine, tangibly
2 embodying a program of instructions executable by the
3 machine to perform method steps for causing one or more
4 functions of this invention.

5 It is noted that the foregoing has outlined some of the
6 more pertinent objects and embodiments of the present
7 invention. This invention may be used for many
8 applications. Thus, although the description is made for
9 particular arrangements and methods, the intent and concept
10 of the invention is suitable and applicable to other
11 arrangements and applications. It will be clear to those
12 skilled in the art that modifications to the disclosed
13 embodiments can be effected without departing from the
14 spirit and scope of the invention. The described
15 embodiments ought to be construed to be merely illustrative
16 of some of the more prominent features and applications of
17 the invention. Other beneficial results can be realized by
18 applying the disclosed invention in a different manner or
19 modifying the invention in ways known to those familiar with
20 the art.